AD_____
(Leave blank)

Award Number:
W81XWH-07-1-0714

TITLE:
Applying A Multi-Voice Speech Recognizer to the BMIST Task

PRINCIPAL INVESTIGATOR:
Gregory J. Gadbois, Ph.D.

CONTRACTING ORGANIZATION: HandHeld Speech
Amesbury MA 01913

REPORT DATE:
October 2008

TYPE OF REPORT:
Final

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
19-10-2008	Final	20 Sep 2007-19 Sep 2008

4. TITLE AND SUBTITLE

Applying A Multi-Voice Speech Recognizer to the BMIST Task

5a. CONTRACT NUMBER

5b. GRANT NUMBER
W81XWH-07-1-0714

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gregory J. Gadbois, Ph.D.

Email: greg@handheldspeech.com

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

HandHeld Speech
Amesbury MA 01913

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Speech recognition topics are explored. An orally driven user interface to form filling was developed. Along the way, unsupervised adaptation methods, noise injection and novel enrollment/"model search" methods were studied. On the strength of the discovered techniques, an application moved from a research topic to a "program of record".

15. SUBJECT TERMS
multi-voice speech recognition, noise-injection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			USAMRMC
U	U	U	UU	19	19b. TELEPHONE NUMBER *(include area code)*

Table of Contents

Introduction:

"Improving hands-free input to the BMIST application" is the broadest description of the work that was to be done in this contract. The specifics were to investigate a number of speech recognition user interfaces strategies (noise injection being a primary one). And the ultimate contract deliverable is an integration of the successful research into a working BMIST application.

An initial observation was that speech recognition does work when the modeling of the speaker's voice in his environment is accurate. A method to obtain accurate speech recognition in real environments would be to dynamically change modeling. If we have a "good set of models" created in a quiet environment, and can track environment changes, we can create a current set of "good models" by injecting the current noise background into the "quiet models", making "noise models".

In the contract proposal we listed four techniques to investigate and combine to make "good" models, they were:

1) Conventional enrollment (reading prompts)
2) Noise injection (creating models from quiet models and a noise sample)
3) Supervised Adaptation
4) Unsupervised Adaptation

This report is a chronology of the work done, reporting on the results of the investigated strategies. And, it is the companion to the final deliverable, documenting the deliverable that is the project goal:

At the conclusion of the contract there will be a final deliverable that is a working BMIST application that incorporates the best strategies found.

An unspecified key detail is: What does "working" mean? Who judges what working means? The answers are Kim Chism and Renee Clerici, the project managers responsible for the BMIST app, who had oversight of this contract. The criteria for "working" was their app would be deployable, that the users would be able to use it, and hopefully prefer using it over any previous method. Their focus is on user frustration, making an app that a user "likes" using. They care little about research per se, except how they can use new technology to enable a "good" solution. These are proper criteria, and this standard drove the research in unexpected directions through the course of the contract.

The contract research goals were in two parts:

1) Research in using speech in the User Interface to create a working solution
2) Research investigating noise injection strategies in particular

We have delivered on these goals. In the body of this report we will give the details of that work.

The ultimate deliverable is the current version of “AHLTA Mobile”. (See the TATRC group for a demonstration -- they have been showing it broadly.) It has generated wide interest, and is perceived to be “a solution”. On the strength of its performance, the AHLTA Mobile application has been recognized as a “Program of Record”, it is not just a research project anymore. In response to its new status, the project has been moved from TATRC to DHIMS. The work done under this contract has been very successful.

Body:

The development group of TATRC had four reasons for this contract. First, the speech recognizer is accurate; it must be accurate to be useful. Second, it takes up little memory, so it can run on smaller platforms like a handheld. Third, they recognized the power that speech recognition in noisy and multi-speaker backgrounds would have. These situations are commonly encountered, and are very difficult for standard speech recognizers. Fourth, they appreciated that both speech recognition and speech user interfaces are very specialized fields and involving a speech UI expert would greatly enhance the likelihood of a successful project.

In our kick-off meeting we set up the project plan. The TATRC group outlined a basic application to use as a vehicle to explore voice interfaces. They set up a joint software project on the Internet so that both a group inhouse at Fort Detrick and HandHeld Speech consultants in Massachusetts, could work on it. Using version control, all the changes were documented. When the changes were substantial, these changes were rolled up into a new version of the application. The team consisted of Mike Vandre and John Pajak at Fort Detrick MD, and Greg

Gadbois, in Amesbury MA. Mr. Vandre and Mr. Pajak provided the test application. Mr. Gadbois implemented all the software dealing directly with speech.

The original app was called “ffw” and already existed in-house at Fort Detrick. It ran on a tablet personal computer and used a touch driven interface. The goal of the ffw application was to document first responder data from the medic (in the field or in the ambulance) so that it could be sent electronically to the field hospital. The medic would fill out a form, by completing different dialog screens by tapping on-screen buttons.
It was originally designed in response to asking prospective end users what they wished they had.

The first four months of work was used to modify the interface of this application from a touch driven interface to one that was both touch and speech driven. The first deliverable “Integrating HHS SDK into BMIST” which was planned to require two man-months was delivered on time.

This required quite a bit of engineering. The c# wrapper over the “hhsDemon” recognition service evolved tremendously. I added new features to the SDK to simplify the implementations needed in the ffw app. Large revisions were made in the underlying libraries.

After four months, the application was functional enough to get subject matter expert (SME) feedback. The goal was to get assessments about the existing functionality and an evaluation of strengths and weaknesses.

The SME evaluations were very interesting. They believed that the speech recognition aspect of the interface worked sufficiently to do the job. However, they didn’t want the hybrid application that required the first responder to look at the screen. Given that the speech recognition was very good, they wanted an application that didn’t require the first responder to look away from the patient to enter data. With the hybrid touch and speech driven application, they had the job of providing care, and a second job of documenting the care, which either could interfere with their ability to provide the care or would have to rely on their memories of what they had done (a source of error). Their goal was to have an application that acted like a smart assistant who was transcribing their data. This new interface would be purely voice driven.

In response to those realizations, a brand new application was conceived. Inhouse it was called TraumaTalk – it is the prototype for AHLTA-Mobile. While we learned a lot in building ffw, it was actually a blessing that we got to throw out the ffw codebase and start from scratch. We were able to redesign, eliminating some basic design flaws that were inherent in the touch-centric design.

TraumaTalk was to be purely voice driven and not require interaction with the machine either by touch or reading displays. The TATRC group conceived the form filling as a dialog, the system prompting with questions (via TTS), the user responding, and depending on answers, the system asking appropriate following questions. The content of a form was conceived as essentially lists of questions and their accompanying responses. The TATRC design group stored the contents of a form as a simple database (in an html format), separating it from a generic machine that loaded such databases. In this manner, it is easy to change the questions and logic flow by just editing an html text file. In fact TraumaTalk is a generic solution to form filling – change the database and you are filling out a different form. Later, the TATRC engineers built tools to create and edit html databases. A software engineer is not needed to create or edit a new form. Using the tools, anyone can create/edit the databases that the generic TraumaTalk machine loads.

The generic machine loads an .html database. When it loads, it creates a state machine with one start state corresponding to the start question. As it loads each question, it generates a speech recognition rule containing the acceptable responses. The questions are very specific; there are typically a small list of single-word or short-phrase responses that are appropriate. At any given time during a dialog, there is one current question and a short list of the things the user might reasonably say that are active in the speech recognizer. The speech recognition problem is very easy. There is no open natural language problem to deal with, where the system would have to recognize unexpected utterances. When the speech recognition problem is kept easy enough, the system works well, even in noise.

One important design flaw of ffw was addressed at the outset, an “undo”.

Speech recognition is inherently statistical and can fail. It is not a disaster to mis-recognize some speech if it is easy

to back out of a mistake and try a different path. A multiple undo/redo is an essential feature for a speech recognition app.
The undo was a feature that was never implemented in ffw (it was proving to be a bear to retrofit). I built an undo capability into TraumaTalk in its basic structures.

I played with our initial version of TraumaTalk. The speech recognition and the interface all worked, but after a bit, the interaction became irritating. The problem was that the interface was designed for a novice user not for an expert. The same questions that were reasonable the first few times became tedious with increased expertise. Having filled out forms a few times, the questions could be predicted. The user had to, wait for utterances to finish such as “What medications were given?” This was inefficient. I wanted a one word question e.g. “Medications?” So I came up with the idea of a short-questions/long-questions option. The idea is, once you have some expertise and know the questions, you only need a one word cue to remind you where you are in the form, and what the current question is.

Next I added a confirmation/no-confirmation option, where it would echo the speech recognition result. The system optionally speaks back what it understood the speaker to say.

I eventually came up with a clarifying understanding of how the user interface app should work by using a thought experiment. Suppose there were two people filling out forms, one had the forms in front of him and was asking questions and writing answers, the other was making all the measurements and saying the answer aloud so that the other could write them. Two people doing this job would start by saying everything, but eventually (if the people got good at it) the dialog would get very terse. Only the minimal important words would be spoken. They would start with long questions, but quickly evolve to single word cues for the questions. When confirming an answer, only the most relevant words would be repeated. In cases where the evolution of states is uniquely determined by the responses, the confirmation is the next question, and no explicit verbal confirmation is needed.

In other cases for example where there is a yes/no answer and the state machine evolves to the same state for both answers, confirmation might take the form of answering yes with “will do” and no with nothing. In the situation where the medic says “blood pressure 140 over 70”, confirmation echoes “140 over 70”. The echoed response is the minimal information and in

particular is the information that the recognizer is most likely to mis-recognize.

Another example of confirmation, suppose the system has just queried “What medications were given?” if the user says “Short questions”, the system confirms by saying “Medications?” The goal of confirmation is to communicate the most information with the fewest words. Repeating the question cue affirms the switch to terse mode and helps him stay on track in the form.

The clarifying ideas are:

1) Make your questions very specific, and the natural responses should be single-words or short-phrases (get rid of the natural language problem).

2) What is the minimal spoken information to indicate questions and confirmations? (maximize information content while minimizing syllables)

3) Always keep in the back of your mind: How would two people who were are either learning the job or becoming expert at it operate?

Support was added to the html database so the terse modes are fully supported.

The resulting speech recognition application is so transparent it is a joy to use. The user soon looses awareness of the speech recognizer, he can focus on making his measurements and doing his or her job. The application becomes a job performance aid rather than an impediment.

The interesting perspective is to compare this user interface to standard ones used for automated answering and routing of phone calls. The telephony solutions are similarly eyes-free applications. However, they are inefficient and unpleasant to use. These systems represent the state of the art in speech recognition interfaces

That perspective makes you realize what a huge step forward we have made with this the user interface of this app. I call this style of solution a “Terse Directed Dialog” (TDD). It is

an evolution on the telephony-like (small vocabulary) solution. But it eliminates the frustration of using those types of systems, while improving efficiency.

I believe it changes the game a lot.

It is important to understand the TDD style solution in the context of current art.

The crux of the difficulty with speech is “How do you communicate to the user, what the speech recognizer is looking for?” There have been two solutions.

1)Telephony type solution, where the speech recognition problem is reduced to a small vocabulary problem

2)Natural Language (pseudo Natural Language) queries where you let the speaker say almost anything.

The telephony solutions are attractive because they work. Current recognizers are good enough that they can reliably do small vocabulary problems even with noisy input. Their downside has been user frustration navigating complicated menu’s.

The Natural Language research is a response to the perceived failure of telephony style solutions. Unfortunately there are a number of hard problems with natural language systems. Chiefly, current recognizers are inadequate to the problem especially with degraded (noisy) input. There is also a problem with extracting meaning and creating the appropriate response. Someday natural language solutions may be attractive, but they are not reliable today.

The Terse Directed Dialog is a solution that works today, even on mobile devices. It relies on how smart people are. It is because people are smart that they find the telephony solutions irritating. After they have navigated a menu once or twice, they become impatient when the system treats them as if they didn’t know what’s going on. Allowing them an accelerated mode solves that problem. It uses their intelligence instead of hindering it. People are much more trainable than computers. A Terse Directed Dialog capitalizes on that fact.

I think the class of problems that are amenable to a TDD solution is huge, and deploys today. I envision systems where after a short getting acquainted time, the user primarily uses

the system in a terse mode. Occasionally, when he is doing something he rarely does, he false back to “Long Questions”, but quickly returns to a terse mode. He only gets verbose information where he needs it. And he is the judge of when that happens.

I believe the idea of a Terse Directed Dialog will have large impact. TraumaTalk itself might do that inside the DOD. TraumaTalk is really a TraumaTalk database and a generic TDD engine. To do other problems in a TDD manner only requires creating a new database, and tools exist to make that job easy.

My guess is that the ideas of a Terse Directed Dialog are not new, that there exist implementations of “expert modes” that share some or all these features. The value of the research is having a clear statement of those principles, then pursuing them systematically in the app. Developing the TDD concept was both a surprise and by far the most important thing done in this contract.

After implementing TDD, we took stock of where we stood, we re-evaluated the application. If you have good models, TraumaTalk is a joy to use. The basic interaction is a joy. The important thing to focus on is the ‘if’ clause – “if you have good models”.

The TraumaTalk TDD engine is using speaker dependent models. A new user went through a conventional enrollment process (repeating prompts) to create models. If enrollment takes too long, people find the process onerous. The strategy employed in the TDD engine was to do minimal, even insufficient enrollment, then let the user force further adaptation during use of the real app. It became clear in some of our testing that some people came out of enrollment with particularly bad models.

As an aside to the topic of noise injection (one of the main topics in the written contract, we will get to noise injection later in this report), if you don’t have good models working in quiet, injecting noise does not fix anything. There is no point in doing any work with noise injection if you can’t first make models that work well in quiet. Noise injection is a secondary solution and relies on first making good “quiet models”.

So the most important thing to improve was “how to create good initial models in quiet”. Principally we needed to do “better” with our conventional enrollment. And we would like to do “better” without seriously adding to the time spent enrolling.

The way our conventional enrollment process worked was, a new user read some prompts, some measurements were made, then a base set of models was warped to fit the measurements. The “warping” is a fairly crude process. Then further prompts are collected and a more refined adaptation process continues. The further the new user’s voice is from the base models, the more extreme is the warping and the resulting models are less consistently good.

The Multi-Voice recognizer enables a new strategy that we decided to investigate. It is quite easy to run multiple sets of models in parallel and see which models fit the voice better. We can do “model search”, we don’t have to do warping with base models that are a bad match to the voice. The idea is we can collect a catalog of base models spanning user voices, do model search, then seed the existing model creation process with a close fit.

We implemented this strategy. Our initial tests suggest this technique will work very well. Currently our implementation is incomplete; we have only two sets of base models. With 10 to 20 sets of models and just a “model search”, we should have a model match that works as good as speaker-independent modeling. With a larger set of base models and a search feeding a creation process, we believe we will reliably make excellent quiet models. It is a future research/engineering problem to obtain the best tradeoffs between performance and enrollment time. The next step in this direction is a significant undertaking and is beyond the scope of this contract.

In the first ffw app we implemented conventional enrollment, supervised and unsupervised adaptation. In the new TDD engine we have implemented an improved enrollment search/creation method and supervised adaptation. The unsupervised adaptation implemented in the earlier software was quite interesting and we have intentions of including such methods in the future. Before we begin the final discussion of noise injection, we would first document the research done on unsupervised adaptation in the ffw app.

The strategy is to watch the behavior of the user to detect when he has corrected a mis-recognition – detect which utterances and phrases were involved and do corrective adaptation. Specifically in the ffw app, the easiest way for the user to correct a mis-recognition was to just to repeat the phrase, hopefully the second time the system would get it right. (When the models are decent, because the second time around the user says the phrase more carefully, typically the system does get it right.) In the software we kept the last couple utterances around and correlated the choice lists of the previous utterance with the current one.

For example, suppose a user says “temperature 101.5” and the recognizer top choice was “temperature 109.5”. (Suppose the second choice was right.) The user repeats saying “temperature 101.5” again. Suppose this time the system gets it right. The software looks back one utterance and finds the current top choice was the high in the choice list of the previous utterance. We detect that we got it wrong the first time but now we know what the right answer is. Probably some of the acoustic models in the word “1” are not so good. We use the second utterance to adapt the models (without asking the user). That’s the basic idea.

We can be a little more sophisticated, if the top choice is the same as the previous top choice, (we see that he is repeating himself, probably we think we are getting it wrong twice in a row) we claim the second choice as the answer and use that for adaptation.

The last variation is that we only mark the utterance as “to be adapted later” and we wait until we are sure he is done (that he doesn’t repeat himself again or force an answer through touch). Only when we believe both that he is happy with the utterance and he has moved on, do we adapt the models.

This strategy was seen to work in the ffw app. It had the effect that “decent” models get sharpened up. When top choices are right all the time, it does nothing to the models. Where there are close calls and occasional mis-recognitions, models are refined and the close calls are separated. It is a good unobtrusive method to evolve “decent” models into “excellent” models. But it does not replace supervised adaptation. If the user repeats the phrase again and again and again, and it never comes up right, the method doesn’t work. You can’t use unsupervised adaptation to make bad models good. It only augments supervised adaptation.

Supervised adaptation is doing adaptation with an utterance where the phrase spoken is known (either because the user was prompted to say the phrase or because he has selected it from a list). A mature app will have both types of methods. Supervised adaptation empowers the user and is crucial as a last resort to fix any problem, unsupervised is desirable because it so unobtrusive.

The only downside for implementing unsupervised adaptation, is that tracking high level behavior is a complication making the app harder to evolve. The codebase is more complicated. When the app is “young” and changing rapidly, it is best to wait before implementing unsupervised adaptation. We plan to add unsupervised adaptation to the TDD engine eventually.

The last part of the research contract was investigating noise injection. The SDK supported some basic noise injection features. We will give a quick intro to the ideas of noise injection next.

When digitized sound is captured, the first thing done is to window and fourier transform it. The power spectrum in frequency domain is further massaged and the result is boiled down to a “speech vector”. The voice model representing a phoneme is a sequence of characteristic speech vectors.

Here are the steps:

$PowerSpectrum = [\ FFT(\ DigitalSound\ )\ ]^2 \rightarrow P = F(\ snd\ )$
$SpeechVector = Massage(\ PowerSpectrum\ ) \rightarrow S = M(\ P\ )$

The process “M” can be inverted. We can take a characteristic speech vector and take it back to a power spectrum. In the power spectrum we can add the signature of the current environmental noise. Then we transform this new power spectrum back to a speech vector.

The process is:

$P_{model} = M^{-1}(S_{model})$
$P_{new} = P_{model} + P_{noise}$
$S_{new} = M(P_{new})$

We developed M and M^{-1} prior to this contract.

Actually there is one caveat that we knew from earlier work. Typically recognition is more accurate when you leave more signature of the model and don't swamp it with the noise spectrum. You want to inject a scaled value of the average noise. There is a parameter 'c' to optimize based on a noise measurement.

$P_{new} = P_{model} + c * P_{noise}$

One of the first things we did in this contract (for noise injection) was to correlate measures of the noise with an optimal value of 'c', a noise scaling. We needed a function that takes a measure of the noise and predicts a good value of the scaling.

In our signal processing there are some power measurements that would be convenient to correlate with optimal injection. We played with a number of them. A good measure was that already existed in our signal processing front end was 512*log(pwr) where pwr is the sum of the power in the voicing part of the spectrum.

This parameter, call it 'k', (with the particular mic gain level we used) ranged from about 40-100 in silence and had voicing peaks of about 2325. We could measure the silence just before and after an utterance, and also the average peak power during the utterance.

In quiet a typical pair was:
(k_s, k_v)
(50, 2325)

In noise a pair might be:
(1300, 2325)

If gain levels are changed, they add a multiplicative factor to the raw pwr:

$k_n = 512*\log(pwr * a) = k_o + 512*\log(a) = k_o + constant$

Define $K = k_v - k_s$; K is a log of the signal to noise ratio. Then $K_n = K_o$ and is independent of a gain setting.

We measured this parameter K, and correlated it with the optimal injection of the noise spectrum. Referring back to the power spectrum formula for injection:
$P_{new} = P_{model} + c * P_{noise}$

We tabulated a function c = f(K), that was the optimal injection value

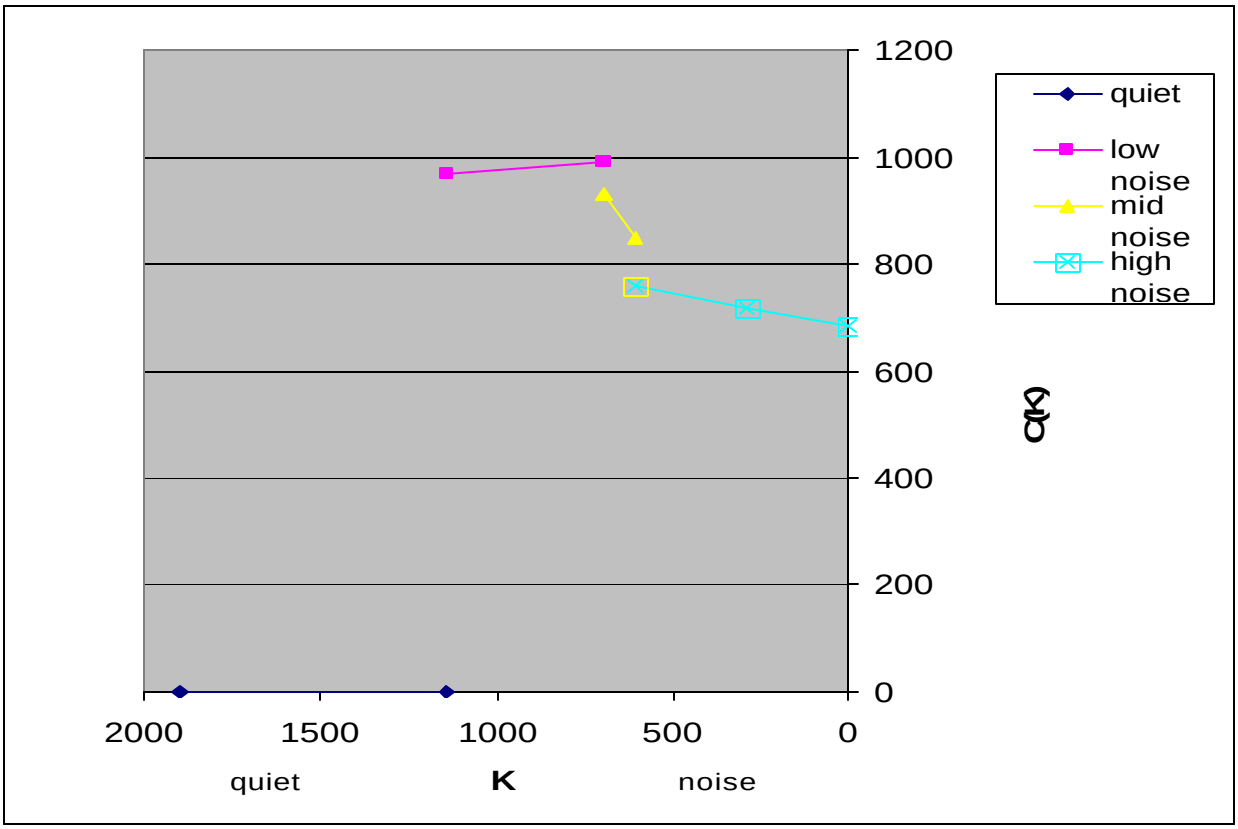
Generally c is a number between 0 and 1. To maintain integer arithmetic, we will use a number between 0 and 1024, then divide by 1024.

c = C/1024;
$P_{new} = P_{model} + (C * P_{noise})/1024;$

C(K) is best describe in a piecewise way.

```
C(K | 1140<K)     = 0;                (quiet)
C(K | 690<K<1140) = 1021 - K*2/45;    (low noise)
C(K | 610<K<690)  = 240 + K;      (mid noise)
C(K | K<610)      = 685 + K/8;        (high noise)
```

(Depending on how hard the recognition problem, if K<300 the accuracy may be unusable.)



One last note, there is a hardware/gain/sanity check: in quiet, K should range from 2230 to 2280 (quiet should be around 50, voicing near 2325). If the hardware performs differently, the data should be scaled to the proper range.

So to inject, we need frames of the background noise, and an utterance to get the signal to noise ratio of voiced-speech to environmental-noise. It would be best if we kept running averages of both. We proceeded to make a simple app with a dialog to collect the background noise and an utterance. (That app will be demonstrated at our final report meeting.) The processes of the dialog are easily collected and could be added as a background process to a normal app, and noise injection could be done automatically. We could continually create models in the background and track the noise environment. We did not apply this logic yet to AHLTA Mobile. Like unsupervised adaptation, it will add complexity to the source code making the app harder to evolve. It is something to be added to a mature app.

Key Research Accomplishment:

1) Developed c# wrappers for the hhsDemon
2) Implemented a Terse Directed Dialog
3) Explored Unsupervised Adaptation
4) Seeding enrollment with “model search”
5) Explored noise measures to control noise injection

These features were significant to the current success – AHLTA-Mobile is now a program of record.

Reportable Outcomes:

1) Showed another HandHeld Speech customer (a group at Naval Undersea Warfare Center, Newport RI) how to do “model search”.
2) Applied for another BAA contract to continue working on AHLTA-Mobile (to further improve rejection and enrollment process).

Conclusion:

The most important result of this contract is that the AHLTA-Mobile app’s user interface is significantly improved. On the strength of the improvements, it is now a program of record.

Of the features driving it, the most important is the Terse Directed Dialog. That concept is a general solution to a wide class of problems.

The next most important thing is “model search”. Model search holds the promise of speaker-indepent recognition that works for everyone.

I believe the unsupervised adaptation strategy is of lesser importance. It is a very nice feature and one likely to be incorporated into a finished app. But it will not have as large an impact as both model search and TDD.

Similarly, noise injection is not as important. There may be some situations where it will make a difference, but the TDD concept marginalizes the number of those cases. The application of Terse Directed Dialogs, reduces the perplexity of the recognition problems – it makes them easy. A recognizer will be accurate, even in noise, when the problem is easy enough.

References:

SBIR contract W81XWH-06-C-0082 awarded 1/16/06. During it we developed the raw noise injection machinery (the “M” and “M^{-1}” operations).

Appendices:

none